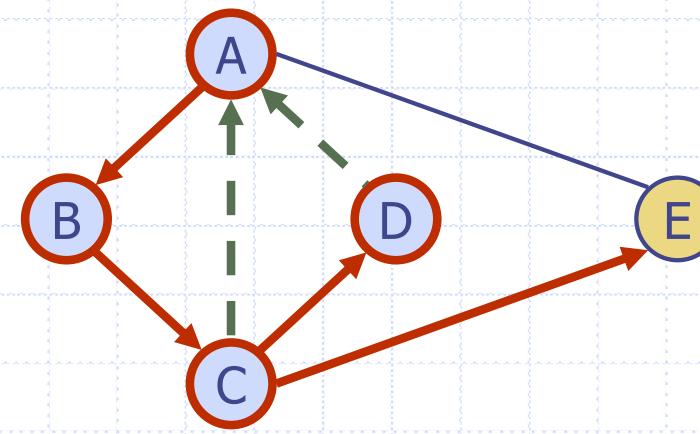


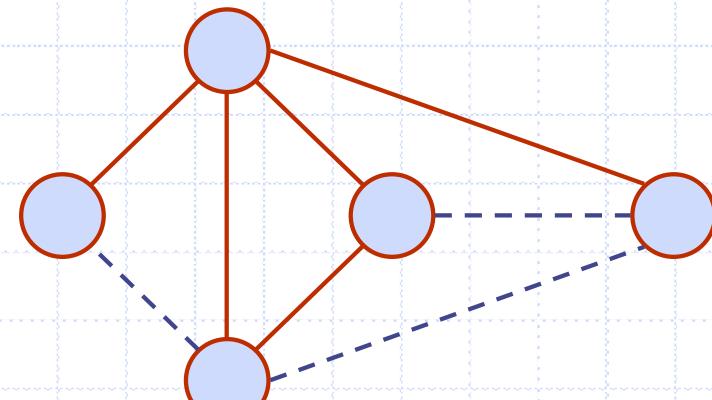
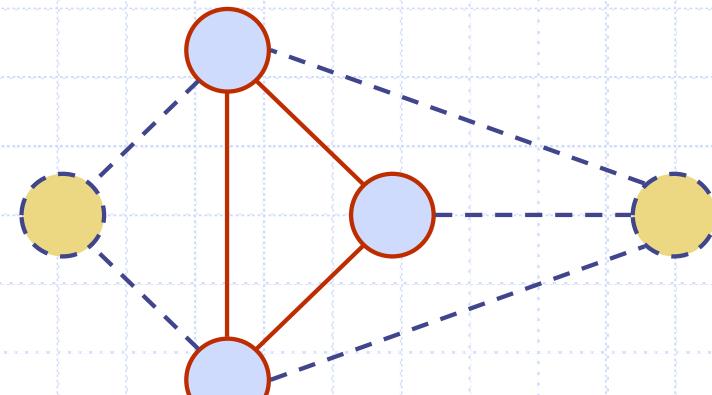
Presentación para usar con el libro de texto **Algorithm Design and Applications**, de MT Goodrich y R. Tamassia, Wiley, 2015

Búsqueda en Profundidad (DFS)



Subgrafos

- Un subgrafo S de un grafo G es un grafo tal que
 - Los vértices de S son un subconjunto de los vértices de G
 - Las aristas de S son un subconjunto de las aristas de G
- Un subgrafo de expansión de G es un subgrafo que contiene todos los vértices de G

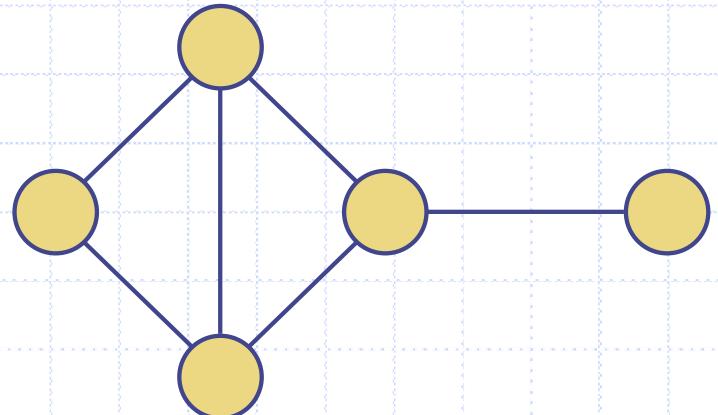


Aplicación: Web Crawler

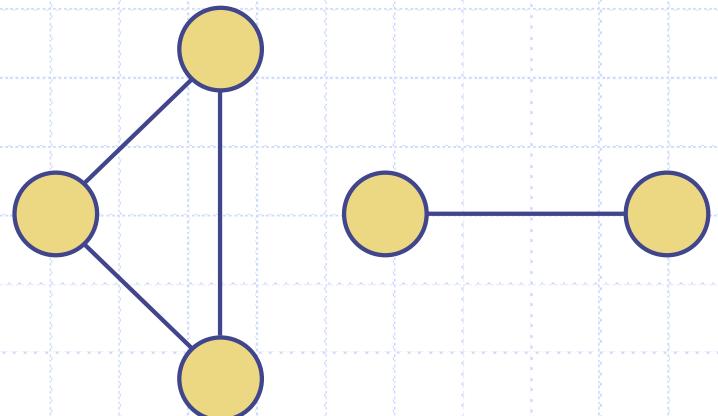
- Un tipo fundamental de operación algorítmica que podríamos desear realizar en un gráfico es **recorrer las aristas y los vértices** del grafo.
- Un **recorrido** es un procedimiento sistemático para explorar un grafo examinando todos sus vértices y aristas.
- Por ejemplo, un **web crawler**, que es la parte de recopilación de datos de un motor de búsqueda, debe explorar un grafo de documentos de hipertexto examinando sus vértices, que son los documentos, y sus aristas, que son los hipervínculos entre documentos.
- Un recorrido es eficiente si visita todos los vértices y aristas en tiempo lineal.

Conectividad

- Un grafo es conexo si hay un camino entre cada par de vértices
- Una componente conexa de un grafo G es un subgrafo conexo maximal de G



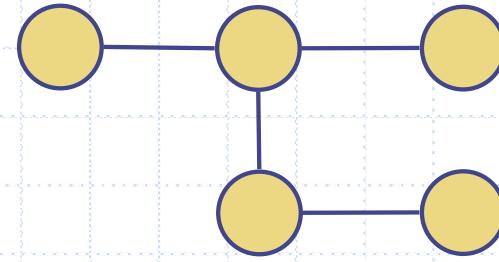
Grafo conectado



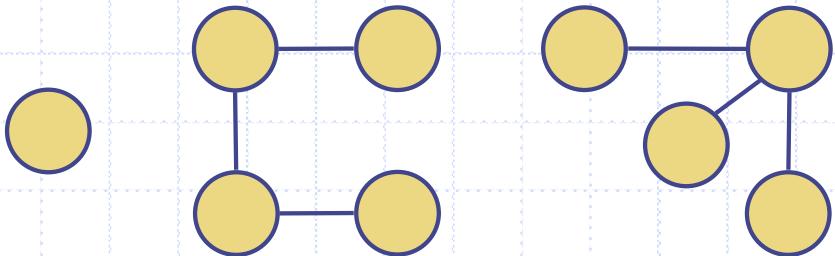
Grafo no conectado con dos componentes conectados

Árboles y Bosques

- Un árbol es un grafo no dirigido T tal que
 - T está conectado
 - T no tiene ciclos
- Esta definición de árbol es diferente a la de un árbol enraizado.
- Un bosque es un grafo no dirigido sin ciclos
- Los componentes conectadas de un bosque son los árboles.



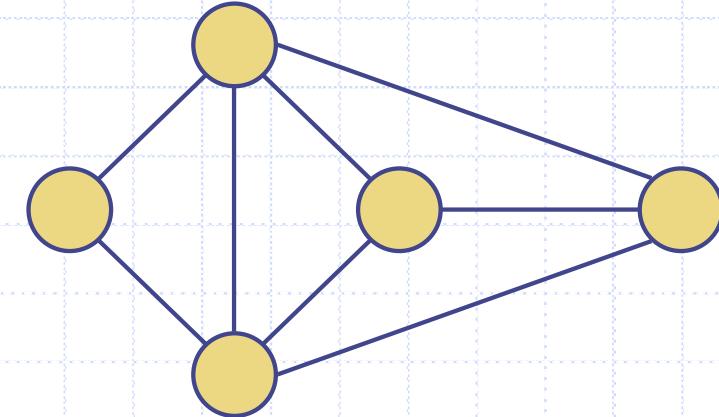
Árbol



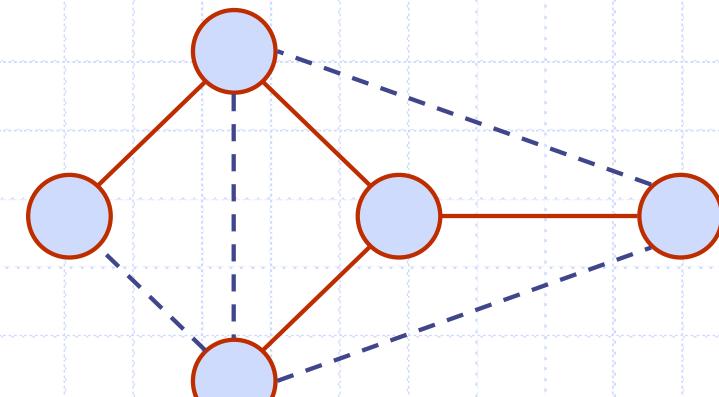
Bosque

Árboles y Bosques de expansión

- Un árbol de expansión de un grafo conexo es un subgrafo de expansión que es un árbol
- Un árbol de expansión no es único a menos que el grafo sea un árbol
- Los árboles de expansión tienen aplicaciones para el diseño de redes de comunicación.
- Un bosque de expansión de un grafo es un subgrafo de expansión que es un bosque



Grafo



Árbol de expansión

Búsqueda en Profundidad

- La búsqueda en profundidad (DFS) es una técnica general para recorrer un grafo
- Un recorrido DFS de un grafo G
 - Visita todos los vértices y aristas de G
 - Determina si G es conexo
 - Calcula las componentes conexas de G
 - Calcula un bosque expansión de G
- DFS en un grafo con n vértices y m aristas toma $O(n + m)$ tiempo
- DFS se puede ampliar aún más para resolver otros problemas de grafos
 - Encontrar un camino entre dos vértices dados
 - Encontrar un ciclo en el grafo
- La búsqueda en profundidad es para los grafos lo que el "*tour de Euler*" es para los árboles binarios

DFS de un vértice

Algorithm $\text{DFS}(G, v)$:

Input: A graph G and a vertex v in G

Output: A labeling of the edges in the connected component of v as discovery edges and back edges, and the vertices in the connected component of v as explored

Label v as explored

for each edge, e , that is incident to v in G **do**

if e is unexplored **then**

 Let w be the end vertex of e opposite from v

if w is unexplored **then**

 Label e as a discovery edge

$\text{DFS}(G, w)$

else

 Label e as a back edge

DFS de un vértice

```
def barrido_profundidad(grafo, vertice):
    """Barrido en profundidad del grafo."""
    while(vertice is not None):
        if(not vertice.visitado):
            vertice.visitado = True
            print(vertice.info)
            adyacentes = vertice.adyacentes.inicio
            while(adyacentes is not None):
                adyacente = buscar_vertice(grafo, adyacentes.destino)
                if(not adyacente.visitado):
                    barrido_profundidad(grafo, adyacente)
                adyacentes = adyacentes.sig
            vertice = vertice.sig
```

Ejemplo



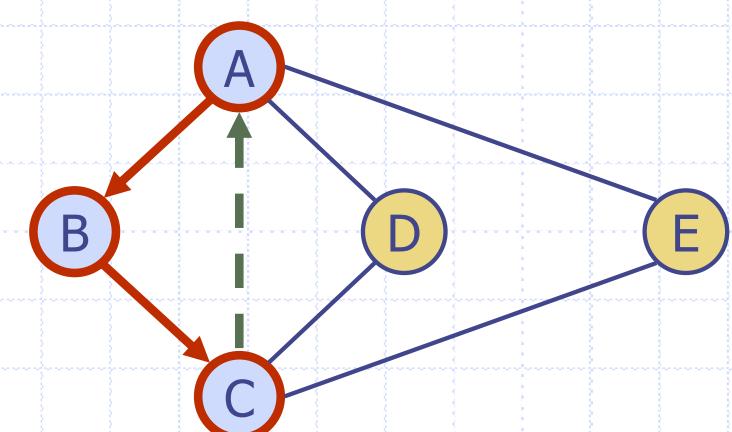
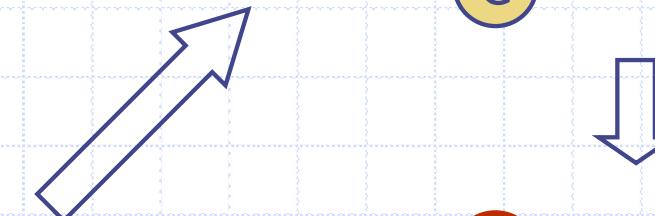
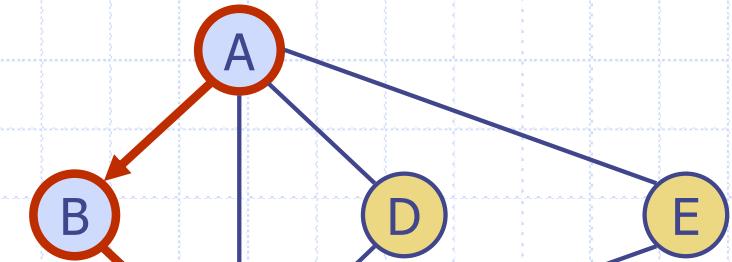
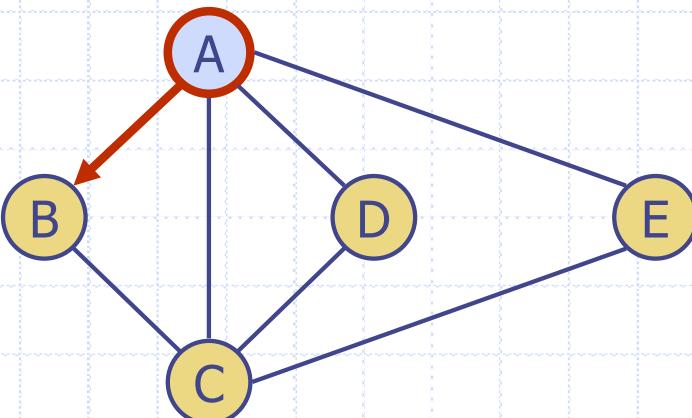
vértice inexplorado

vértice visitado

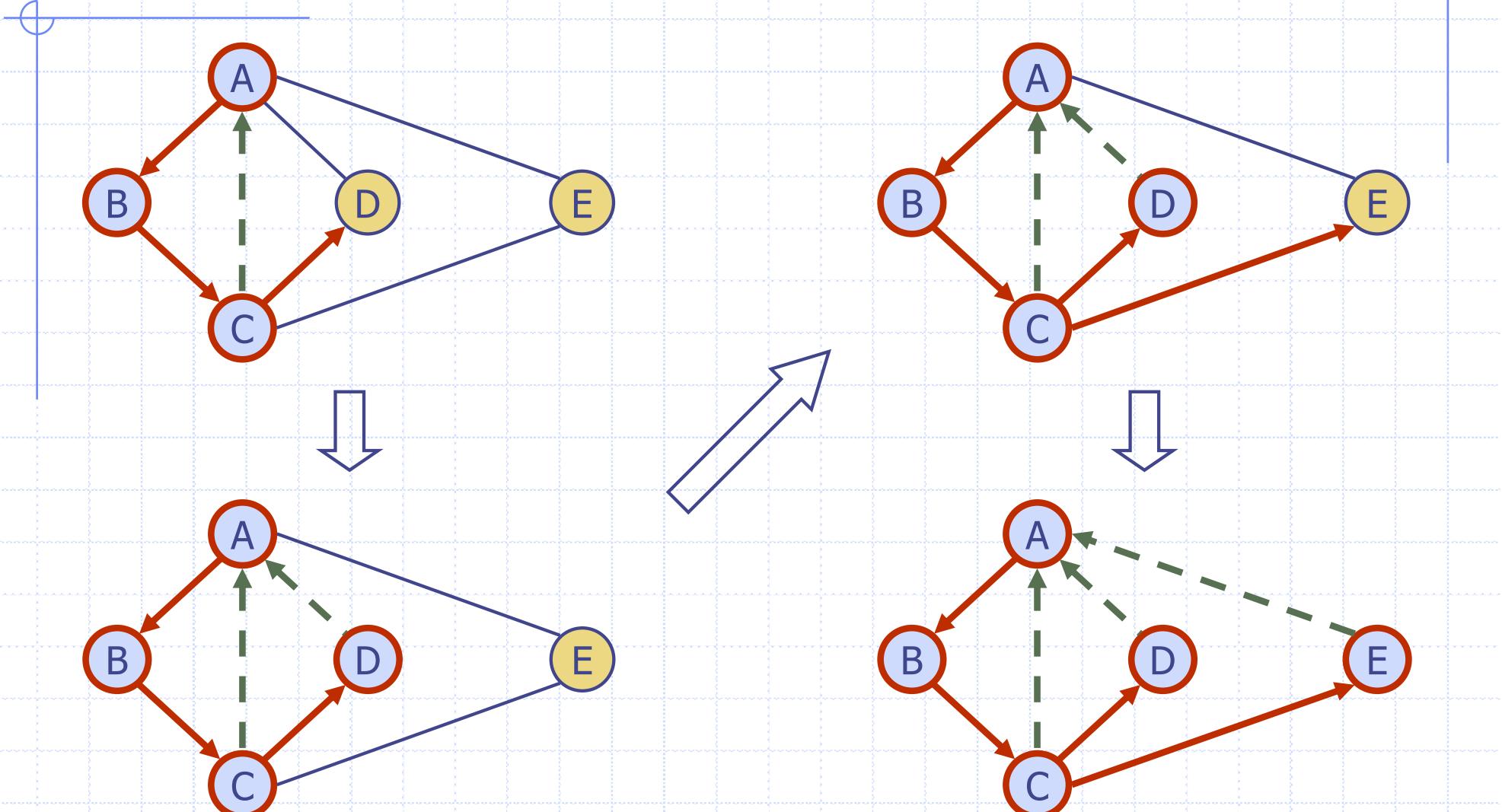
arista inexplorada

arista visitada

—→ arista de regreso

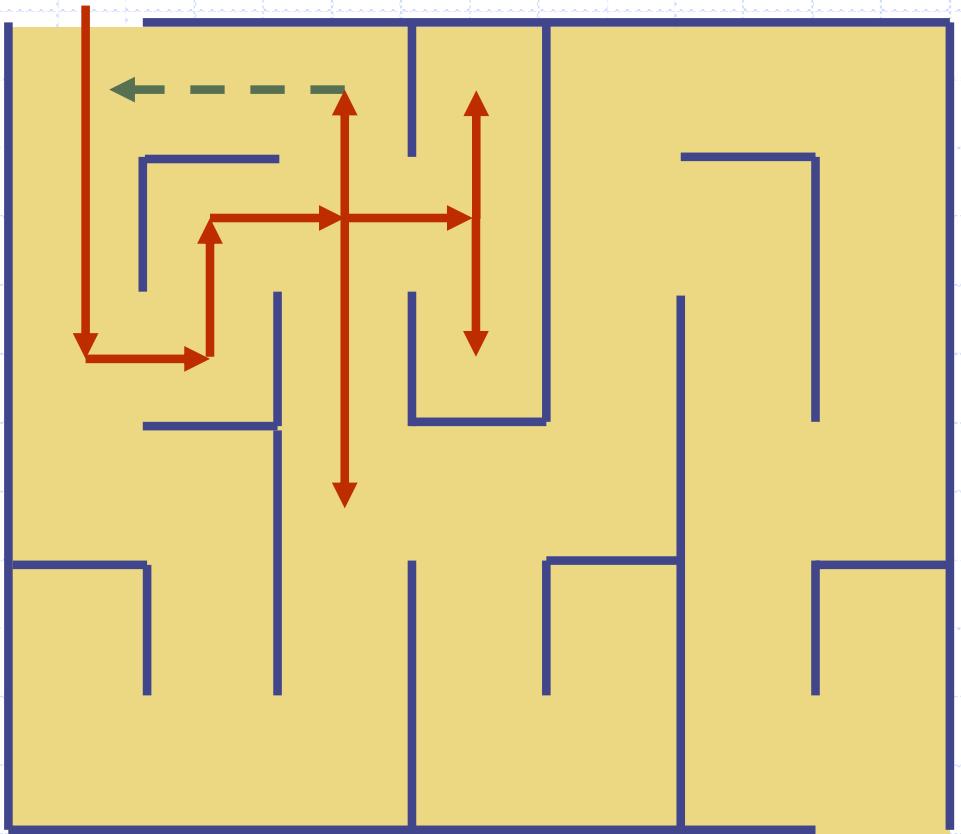


Ejemplo (Cont.)



DFS y Recorrido de Laberintos

- El algoritmo DFS es similar a una estrategia clásica para explorar un laberinto
 - Marcamos cada intersección, esquina y callejón sin salida (vértice) visitado
 - Marcamos cada corredor (arista) atravesado
 - Hacemos un seguimiento del camino de regreso a la entrada (vértice de inicio) por medio de una cuerda (pila de recursión)



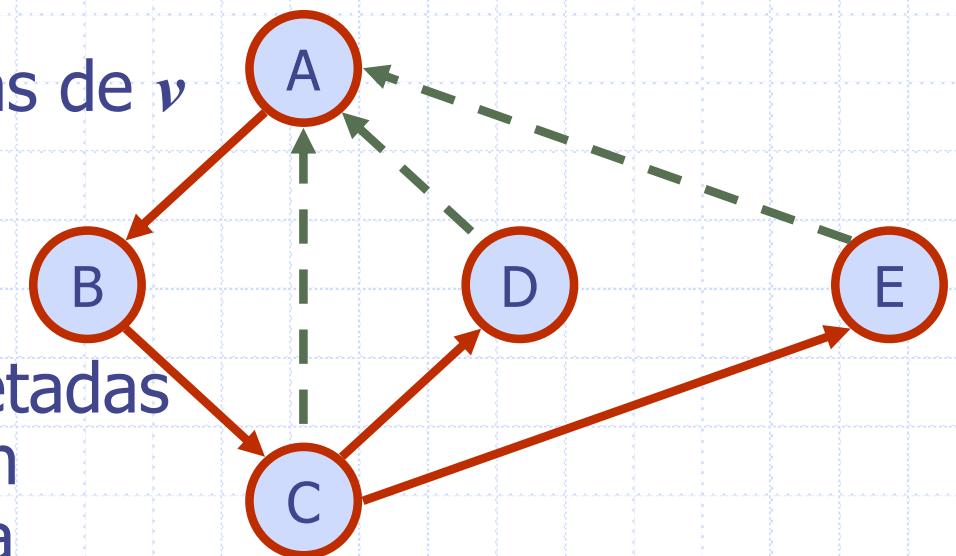
Propiedades de DFS

Propiedad 1

$DFS(G, v)$ visita todos los vértices y aristas de la componentes conectadas de v

Propiedad 2

Las aristas visitadas etiquetadas por $DFS(G, v)$ forman un árbol de expansión de la componente conectada de v



El algoritmo general DFS

- Realice un DFS de cada vértice inexplorado:

Algorithm $\text{DFS}(G)$:

Input: A graph G

Output: A labeling of the vertices in each connected component of G as explored

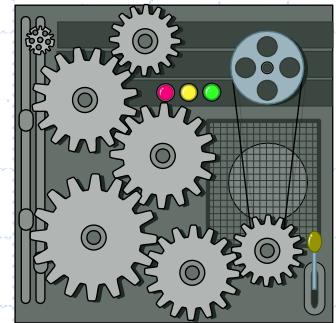
Initially label each vertex in v as unexplored

for each vertex, v , in G **do**

if v is unexplored **then**

$\text{DFS}(G, v)$

Análisis de DFS



- Establecer/obtener una etiqueta de vértice/arista lleva $O(1)$ tiempo (constante)
- Cada vértice está etiquetado dos veces
 - una vez como INEXPLORADO
 - una vez como VISITADO
- Cada arista está etiquetada dos veces
 - una vez como INEXPLORADO
 - una vez como VISITADA o RETORNO
- El método **aristasIncidentes** se llama una vez para cada vértice
- DFS se ejecuta en tiempo $O(n + m)$ siempre que el grafo esté representado por la estructura de lista de adyacencia
 - Recuerde que $\sum_v \deg(v) = 2m$

Búsqueda de Caminos



- Podemos especializar el algoritmo DFS para encontrar un camino entre dos vértices u y z usando el patrón del método
- Llamamos $\text{DFS}(G, u)$ con u como el vértice de inicio
- Usamos una pila S para realizar un seguimiento de la ruta entre el vértice inicial y el vértice actual
- Tan pronto como se encuentra el vértice de destino z , devolvemos el contenido de la pila como ruta

Algoritmo ***caminoDFS*** (G, v, z)

setLabel(v, VISITADO)

S.push(v)

si $v = z$

devolver *S.elementos()*

para todo $e \in G.\text{aristasIncidentes}(v)$

si *getLabel(e)* = SIN EXPLORAR

$w \leftarrow \text{opuesto}(v, e)$

si *getLabel(w)* = SIN EXPLORAR

setLabel(e, VISITADA)

S.push(e)

caminoDFS(G, w, z)

S.pop(e)

else

setLabel(e, REGRESO)

S.pop(v)

Búsqueda de Ciclos



- Podemos especializar el algoritmo DFS para encontrar un ciclo simple usando el patrón del método
- Usamos una pila S para realizar un seguimiento de la ruta entre el vértice inicial y el vértice actual
- Tan pronto como se encuentra una arista de regreso (v, w) , devolvemos el ciclo como la parte de la pila desde la parte superior hasta el vértice w

```
Algoritmo cicloDFS( $G, v, z$ )
    setLabel ( $v, VISITADO$ )
     $S.push ( v )$ 
    para todo  $e \in G.aristaIncidentes( v )$ 
        if getLabel(  $e$  ) = SIN EXPLORAR
             $w \leftarrow$  opuesto(  $v, e$  )
             $S.push ( e )$ 
            if getLabel(  $w$  ) = SIN EXPLORAR
                setLabel(  $e, VISITADA$  )
                caminoDFS (  $G, w, z$  )
                 $S.pop ( e )$ 
            else
                 $T \leftarrow$  nueva pila vacía
                repetir
                     $o \leftarrow S.pop ()$ 
                     $T.push ( o )$ 
                hasta que  $o = w$ 
                devolver T.elementos()
     $S.pop ( v )$ 
```